

SEAL's operating manual: a Spatially-bounded Economic Agent-based Lab

Bernardo Alves Furtado, Isaque Daniel Rocha Eberhardt, and Alexandre Messa

Abstract—This text reports in detail how SEAL, a modeling framework for the economy based on individual agents and firms, works. Thus, it aims to be an usage manual for those wishing to use SEAL or SEAL's results. As a reference work, theoretical and research studies are only cited. SEAL is thought as a Lab that enables the simulation of the economy with spatially bounded microeconomic-based computational agents. Part of the novelty of SEAL comes from the possibility of simulating the economy in space and the instantiation of different public offices, i.e. government institutions, with embedded markets and actual data. SEAL is designed for Public Policy analysis, specifically those related to Public Finance, Taxes and Real Estate.

Index Terms—Agent-based modeling, public policy, Python, computational economics, municipalities, public finance.



1 INTRODUCTION

THIS text reports in detail how SEAL works.¹ Thus, it aims to be a reference for those wishing to use SEAL as a model, use SEAL's outputs, study its details, or work as a SEAL's collaborator. As a manual, the focus of the text is not on a specific research question or a literature review. Thus, the accompanying literature is not discussed in the report. We direct the interested reader to other sources [1]–[6]. Rather, here we describe the framework, the Lab, that we hope may be used for a number of applications. Henceforth, we have applied it to [7], [8]. This report follows the ODD protocol recommendations of Grimm and colleagues [9]. However, we aim at being more specific, describing in detail the components of the model.

We refer to SEAL as a Lab or a framework that enables the simulation of the economy, its markets, considering spatially bounded computational agents. SEAL simulates the economy in space, its markets and government institutions. SEAL is specifically designed to

study Public Policy, with an emphasis on Public Finance, Taxes and Real Estate.

This claim of usability derives directly from the available modeling elements present in the model. The Lab comprises Citizens, organized as families, living and moving among households, traveling to firms both of which: households and firms, are spatially geocoded within actual municipalities' boundaries. Citizens are born, age and die and they are initiated in the model according to official statistics. Municipalities' government is also present. Finally, interaction amongst agents happens in three markets: labor, real estate and goods. All those agents are operated in an agent-based model simulated in Python 3.4.4.

Besides this introduction, this report includes a simple statement of purpose (section 2), a description of the process overview and scheduling (section 3), a full description of the agents, its methods, the markets (section 4) and the different possibilities of simulations to run (section 5). Section 6 describes all the necessary data input, parameters setting and demographic and spatial information necessary to run SEAL for different configurations (section 6). Section 7 delineates the outputs of SEAL (section 7). This report concludes (section 8) with the description of the design concepts, following [9], and the possibilities we see

• B. Furtado, I. Eberhardt and A. Silva are with the Institute for Applied Economic Research, Government of Brazil, Brasília, Brazil. B. Furtado is also a researcher at CNPq
E-mail: bernardo.furtado@ipea.gov.br

1. This applies to SEAL version 2.0 as of September 2016.

for this framework (section 9).

2 PURPOSE

SEAL was designed specifically to tackle public policy concerns. A main justification can be found in [10]. As such, SEAL provides an economic-based framework to answer a number of questions. A proposed list is available in [7], section 4. Illustratively, we could cite: general evaluation of tax impacts; economic impacts on mobility; general impacts of agents' decisions; impacts of changes in government spatial outreach, to name a few.

3 PROCESS OVERVIEW AND SCHEDULE

SEAL is setup so that the modeler can run a single test or multiple runs, in order to make sure the results are consistent (see section 5). When running a single run, the modeler should simply type: `python main.py` in a `python 3.X` interpreter (Figure 1). However, it would be wise to set a number of parameters first (see section 6) and make sure all the necessary libraries are installed.

3.1 Time_iteration module

SEAL is organized around a discrete time frame. As such, time is divided into days, months, quarters and years. It is possible to insert periodical activities, for any of the agents in the model, at any one of these periods.

The first function that runs before the actual `time_iteration` is `have_a_go`.

`Have_a_go` initiates `my_simulation` as an instance of `time_iteration.TimeControl` class. Then it runs up to the total number of days setup for the full simulation. Typically, since we only consider working days, months have 21 days, thus, we run for 5.040 days (which is equivalent to 20 years).

3.1.1 Daily activities

Daily activities are restricted to production updates. On Day 0 of the simulation, however, some setup is put in place before actual simulation begins. This includes the single time a

product is created ² and a first round of labor market matching up, so that firms may have employees and be able to produce during the first month of simulation.

3.1.2 Monthly activities

In the beginning of every month, the demographics dynamic is updated. Given the citizen's month of birth, the probabilities of her death and of her giving birth (see details at section 4.7) are calculated. Then, firms pay salaries (workers receive their salaries before they can go shopping).

Families distribute their money equally among their members. Then, each member acquires his consumption goods, and immediately consumes them (see details at goods market (see section 4.6.1). Governments collect taxes from consumption. In turn, firms update or maintain their prices according to their inventory levels (see section 4.5). Finally, each government municipality spends the collected taxes on public goods. The region quality of life (QLI) is improved by the same amount as the government spending per capita. However, a `TREASURE_INTO_SERVICES` parameter adjusts the numbers so that QLIs remain close to a 0 to 1 indicator.

Before the end of the month, the labor market (see section 4.6.1) is processed, followed by one run of the real estate market (see section 4.6.3). Finally, statistics and outputs are calculated and saved (section 7). Presently, quarters and years are used exclusively for statistics computation.

Four other functions are located at `time_iteration` for importing or usage reasons. However, they pertain to the labor market and the equalization of resources among family members.

In short, activities run every month are subsequently:

- 1) Record month
- 2) Process demographics (aging, births, deaths).
- 3) Firms make payments.

2. The structure of the Lab is compatible with the idea that companies may create more products with different characteristics during simulation. The current configuration does not differentiate among products, which are homogeneous.

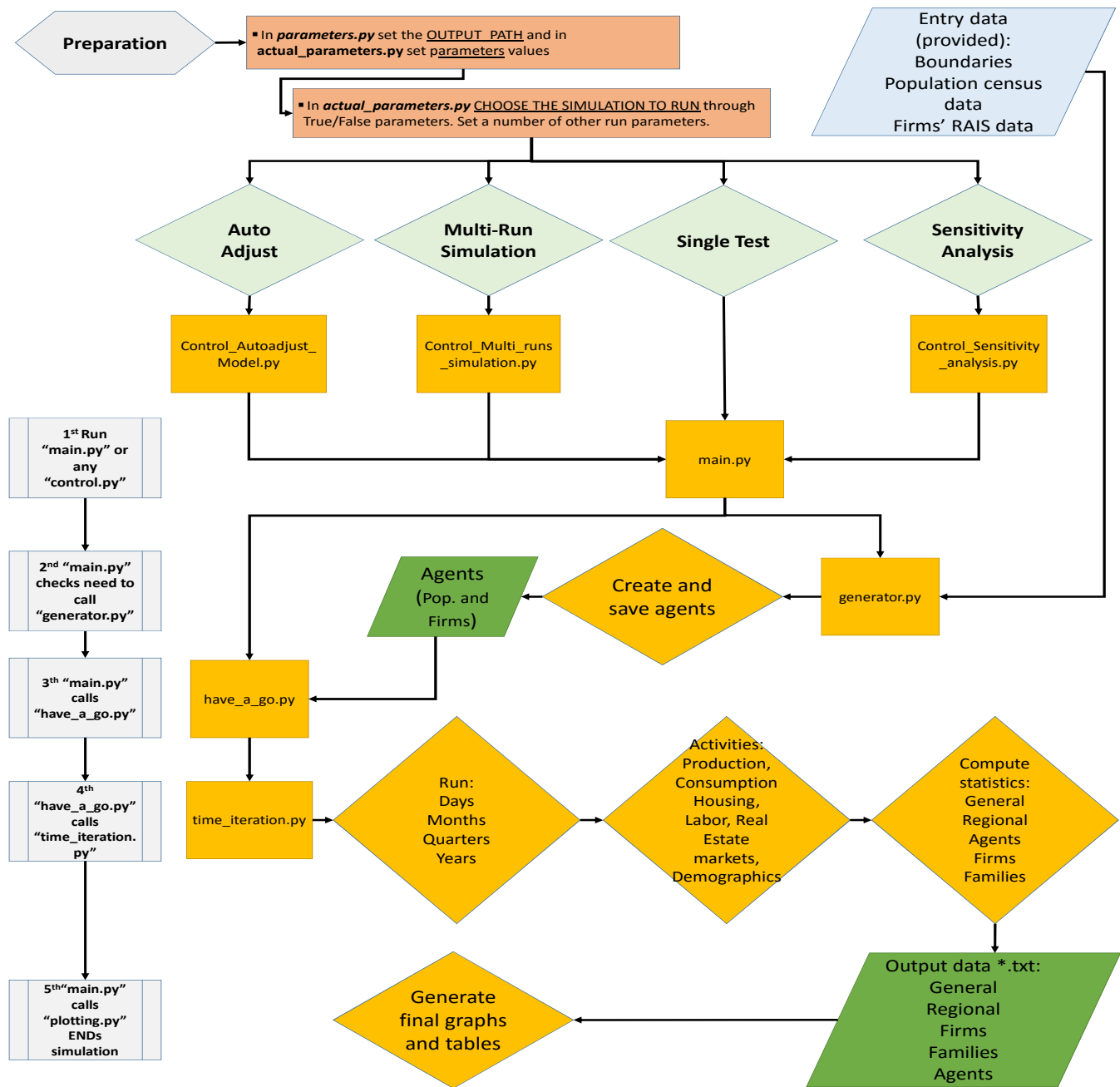


Fig. 1. SEAL flowchart for the major processes and `python` modules. The grey objects are definitions or data to start running a simulation. Population data come from Brazilian demographics census; number and size of firms from RAIS. The orange objects represent decisions to be made while running the model. Rectangles are parameters definitions and the lozenges represent the kind of simulation chosen. The gold rectangles represent the most important modules, which control the model and run the processes. Green parallelograms represent output data (graphs, maps, txt and csv files).

- 4) Families redistribute their own cash within members.
- 5) Family members consume (Goods market).
- 6) Governments collect taxes.
- 7) Governments spend the collected taxes on life quality improvement (QLI).
- 8) Firms calculate profits and update prices.
- 9) Labor market is processed.
- 10) Real estate market is processed.
- 11) Statistics and output are processed.

4 ACTUAL ENTITIES AND PROCESSES

This section aims at detailing each model agent variables and methods. All five following agent types are treated as classes and instantiated once (see section 6.3) before running the simulation.

4.1 Citizens

Agents have the following variables:

- fixed: `id`, `gender`, month of birth, `qualification`, `family_id`.
- variable: `age`, `money` (amount owned at any given moment), `savings`, `firm_id`, `utility` (an indicator of cumulative amount consumed), `address` (`osgeo`, `ogr`), `distance`, `region_id`.

When instantiated, agents are given an `id` (in successive order of generation), a `gender`, an initial `age`, a fixed `qualification` and some initial `money`. The number of agents, their `gender` and their `age` come from official data (see section 6.3). `Qualification` is drawn from a gamma distribution with $\alpha = \beta = 3$. Thus, it is independent of `age` and fixed throughout the model. `Money` is drawn from a uniform distribution in (20, 40).

Agents have a number of methods. The usual `get()` methods are used to return variables values, to check conditions related to `age` (such as `is_minor` and `is_retired`), and update variables (such as `update_age` and `update_money` given a certain amount).

A number of methods are related to family business. `set_family`, which then never changes, `get_family_id` and a check to see if

the agent belongs to `family`. Address and consequent region membership change according to the changes the family does. Every time a family changes addresses, values are updated at the agent level.

Methods related to the workplace include setting and getting workplace and a check to see if the agent is `employed`. When `firm_id` is `None`, then either agent is unemployed or is out of working age. The variable `distance` always refers to the distance between current household where the agent resides and current firm, when employed. Otherwise, it is set to `None`.

Each agent also has a method that calculates distance from his or her current house location to prospective and current place of employment. Information that is printed and outputted in TXT format individually for each agent includes: `id`, `gender`, `qualification`, `age`, `money`, `firm_id`, `utility`, `address`, and `region_id`.

4.2 Citizens' families

Families have a single `id` variable. They also hold values that are constantly updated for `balance`, `savings`, `household_id` and `region_id` (initiated at `None`), `house_price`, `address` and `house`. `House` is a variable that actually contains the full instance of a household. It is useful to access the house price, its address and region. This scheme is necessary because families move from house to house, given the endogenous changes of the model as a whole.

Finally, an important variable of families is an inventory `members` (actually, a Python dictionary) that contains all the instances of the families' members. Related proper methods are `is_member` and `num_members`, which report the family size.

The methods in Families include typical `get` and `set` ones; an `add_agent`, which brings the agent instance into `members` and sets the `family_id` inside the agent instance. A counterpart method is that of `death_member`, which removes the member from the family (at the death event). Families also have access to `sum_balance`, `sum_savings` and

`update_balance`, which is a method that divides a received amount equally among family members.

Other two important pair of methods are `assign_house` which updates `household_id`, `region_id`, the house itself, and `empty_house` (which makes all those variables into `None`, and also calls `empty()` within the house class). `set_address` updates `address` and `region_id` for all members within the family.

A relevant function `consume()` is hosted at the level of the family, but it will be described in the Goods and Services Market, section 4.6.1.

Finally, families calculate internally the proportion of employed members as well as average utility.

4.3 Households

Households are simple, fixed class objects that contain mostly get and set methods. Fixed variables include: `house_id`, `address`, `size`, `region_id`, `quality`, all of which are set at the beginning via the generator module (section 6.3). Among those methods, we can list: `get_family_id`, `update_ownership`, `is_occupied`, `add_family`, and `empty` which changes `family_id` into `None`.

Non-fixed variables include `price`, `family_id` and `ownership` so that both the current family living on a given household as well as its owner can change dynamically.

These processes of changing ownership and residence both happen at the real estate market (section 4.6.3). Price is updated through the relation `size times quality times a region Quality of Life Index` (the latter is updated at the Government module, according to the amount of taxes collected).

4.4 Government

This class represents the individual administrative-political unit basically, the municipalities. It is generated based on shapefile objects (that contain the geometric features of the boundaries), using OSGEO objects.

It is the REGION where taxes are applied and the REGION where taxes collected are spent in order to increase the quality of life. The class also contains self-calculated information including firms and citizens within its territory.

In detail.

The generation of the Govern class object needs to be read as ogr from OSGEO. This implies that an `address_envelope`, which is read from `geometry().GetEnvelope()` is possible as an `__init__` method. Addresses also comes from `geometry()`.

Apart from the geometric features, Govern has a name which is extracted from the available fields associated in the shapefile using `GetFieldAsString(0)` and an ID which also comes from a field `GetField(0)`.

Other variables include `index`, which is the current value of the Quality of Life Index³, `treasure`, the current amount of available cash, the `region_gdp`, which is set at statistics (section 7.1), its population (`pop`), and a calculated `total_commute`.

The method to calculate the population receives all the families as input and for those families in which `region_id` is the same of the current region, the members of such family are added up.

The government spending (that is, the amount of collected taxes) is directly converted in increase in the quality of life index. In order to keep values nearly compatible with empirical data, `treasure` is normalized dividing it by population, and multiplying the result by a reducing value of `TREASURE_INTOSERVICES = 0.0005`.

4.5 Firms

Firms are generated at the start of the simulation and remain fixed afterwards. Thus, `firm_id`, `address` and `region_id` are fixed variables. Besides, at the beginning of the simulation, each firm has a small amount of cash that is stored at `total_balance`.

However, the number of employed workers, their quality, and the price of the firms' product vary through time. Variables used to follow the

3. For simulation purposes, this index is initiated as the value of the municipal Human Development Index in 2000

firms dynamics include: `last_qtr_balance`, updated every three months; `profit`⁴ whether current `total_balance` is above `last_qtr_balance - amount_sold` that saves the cumulative number of products sold; and `amount_produced`. The usual `get` and `set` methods allow access to the values of the firms' variables.

The firms also contain a dictionary of employees with all of its employees at any given time. As an object oriented programming paradigm, the dictionary carries the full instances of the employed agents, thus providing access to all of the agents' information.

Prices can go up, down or remain the same. Prices remain unchanged when either the number of employees or the inventory are zero.

When a firms inventory (Q) is above a given parameter (K) (`QUANTITY_TO_CHANGE_PRICES`), then the price of its product is reduced in proportion to the `MARKUP` parameter. The opposite happens when quantity goes below that same level, and they increase proportional to the `MARKUP` value.

$$price_t = \begin{cases} price_{t-1} * (1 + markup), & \text{if } Q < K \\ price_{t-1} * (1 - markup), & \text{otherwise} \end{cases} \quad (1)$$

`Sale` is the method that operationalizes the goods market. It is detailed in section 4.6.1.

Other methods pertaining the functioning of the firm are related to employees' business. Those include `add_employee`, which not only includes the agent in the employee dictionary, but also passes the `firm_id` to the agent (who can then store it and thus change his status to employed). The `obit` method deletes the employee from the firms' dictionary when he or she passes (see `demographics` module). When `fire` is called, the employee is also removed from the dictionary of employees of the firm and his `firm_id` is set back to `None`.

A relevant role of the firm is to control staff at all times.

A method to `add_employee` makes sure to register the agent within a firm's dictionary employees. It also adds `firm_id` to the

agent using method `set_workplace`. When an employee dies and is withdrawn from the simulation, he is deleted from the firm's team with `obit`. When called, `fire` chooses a (single) random employee and deletes him or her from the staff and also sets workplace to `None`. Other methods related to employment include `a_check`, `is_worker` and `num_employees` (which returns the size of current staff).

Payment of employees is such that when the firm is having profits and cashflow is positive, profits are distributed as salary/bonuses using individual productivity criteria. Otherwise, salary is based on productivity times a unitary price.

Payment of employees is calculated in a two-stage process. When the firm is distributing profits, a `get_wage_base` is defined, such that the base salary is the unit plus the percentage of profits over current available cash. So that if the company has a current cashflow of 1,000 and profit was 200, then base salary is incremented by a 20% margin at 1.20. Otherwise, base salary remains at unit. Then, payment is made exactly according to production. That is, each employee produces the quantity equivalent to qualification exponentiated by a productivity parameter α .

Thus, salary is the unit (or unit plus profit distribution margin) times days of production (21) times qualification exponentiated by α .

$$salary_t = \frac{profit_{t-1}}{cashflow_{t-1}} * qualification^\alpha \quad (2)$$

4.5.1 Products

Another dictionary is `inventory` which contains the list of the firm's products. At the current implementation, there is only a single product. However, code is implemented so that other products could be easily added. In order to save and control the IDs of possible products, there is also a `product_index` variable, which starts at 0.

A special method, `create_product`, is used only once, before the simulation actually begins. Products contain information on `price`, originally set at 1, and `quantity`, obviously, initially set at 0. They are also produced as a class, stored at `products` module.

4. Profit starts at 1 in order to guarantee an opening positive value.

`update_product_quantity` is actually the method that implements the production function. Production is based on the quantity and quality of the employees. Given that any employee has a given level of qualification, the quantity produced is each employee's qualification raised to an alpha parameter that indicates the productivity level. Thus:

$$Q_{it} = \sum_{j=1}^J j_i^\alpha \quad (3)$$

where Q_i represents the quantity produced by firm i , j_i , each employee qualification, and α , the productivity parameter. Production only occurs if the firm has a positive number of employees, positive monetary funds and at least one registered product at inventory.

4.6 Markets

There are three markets at the current implementation of the model. The operation of goods and services is distributed among methods within the agents and the firms, and is monthly called for every agent at `time_iteration`. In turn, the labor market has a module of its own, named `communications` indeed, that market was thought out as an announcement board where companies and potential employees meet. Finally, the real estate market also has a module of its own, called `housing_market`.

4.6.1 Goods and services

The function `consume` is located within each family with the following steps. First, the family needs to have a positive amount of cash in order to enter the market. If that amount is below the unit, a random number between 0 and the available amount is chosen to be spent.

If, on the other hand, that amount is above the unit, the chosen consumption is a percentage of the available amount of money (M). Such percentage is given by a number drawn from a beta distribution, so that the average is given by the modeler chosen β parameter, i.e. an average propensity to consume

$$to_spend \sim \begin{cases} U(0, 1) * M, & \text{if } M < 1 \\ Beta(1, \frac{1-\beta}{\beta}) * M, & \text{otherwise} \end{cases} \quad (4)$$

Once the decision of the amount to be spent is made, the agent checks another parameter called `SIZE_MARKET`. This parameter determines the number of firms the agent makes contact before doing the consumption decision.

Third, among the contacted firms, the agent chooses to buy either from the cheapest firm, or the closest one. This decision is random and equally weighted with probability 0.5.

Once those decisions have been made, consumption is straightforward, given that products are homogeneous.

If the firm can sell the full agent demand, according to his full amount to spend, the sale occurs. Otherwise, the firm sells as much quantity as it can provide, and returns monetary change for the quantity it cannot provide.

At each month, the cash that the agent decides not to spend goes to a savings account. This account enables the family to enter the real estate market.

`Sales` are located within the firm agents. A simple check is made to see whether the amount available from the consumer is positive. Given the original structure that allowed for more than one product, the amount to spend is distributed equally among the products available. However, for this implementation, there is a single product available and the entire amount is directed to such a product.

The first operation within the firm is to transform the cash amount into product quantity, given current price. The resulting quantity is deduced from the firm's inventory and taxes are deduced from the amount paid. The firm's balance is increased by the amount bought, and the treasure of the region where the firm is located is increased by the corresponding tax collected (that is, given the tax rate and the amount spent). The firm registers the amount sold for statistics purposes.

4.6.2 Labor

At the end of every month, the labor market is initiated from `time_iteration` that calls `hire_fire` and `look_for_jobs`: the first function fills in a list with the firms that are hiring; the second one compiles a list of able and currently unemployed workers.

The entrance of the firm in the labor market is determined by a parameter (set by the modeler), at the parameters module, called `LABOUR_MARKET`. The larger this parameter, less often the firm enters the labor market. If it is set, for instance, at 0.75, it means that, at each month, the firm has a 25% chance of entering the market on average, firms would then enter the labor market once every four months.

At the labor market, the firm decides to hire if its profits are positive (profits are calculated based on cash flow, having last quarter as reference). Otherwise, the firm fires one employee. That is, if the firm has a negative result since last quarter and the strategy is to enter the market, then the firm makes one employee redundant. Decision on the employee is made randomly. That process guarantees a dynamic labor market.

On the side of likely work seekers, citizens enter the market every month when they fulfill all of the three following conditions: (a) being currently unemployed, (b) being of age 16 or older, (c) being active in the market (up to 69 years old).

The module `communications`, that makes the matching among firms and employees, has a class called `Posting`. There are two lists, one of candidates and one of available postings. They contain all the firms and citizens. The method that makes the match is `assign_post`.

First, both lists are sorted: the firms offering posts are sorted from those paying the highest base salaries; and the agents looking for jobs are sorted by years of qualification (and, implicitly, productivity). Then, the firm paying the highest salary chooses first: the match is made randomly with a 0.5 probability for the candidate living the closest to the firm or the most qualified in the list. While there are firms offering opened positions and candidates interested, the match goes on. Every month both lists are emptied and the process restarts.

4.6.3 Real estate

The houses prices are updated according to current Quality of Life Indexes (QLI) of their respective regions (municipalities). Thus, every time the municipality collects taxes and

spends in public goods, the offering prices of households increase accordingly. In order to make its value comparable to actual Human Development Index for Municipalities, the QLI is normalized by multiplying it by a parameter (k) `TREASURE_INT0_SERVICE` (typically set at 0.0005). Further, QLI is weighted by the change in population (N).

$$QLI_t = \frac{QLI_{t-1} * N_{t-1}}{N_t} + \frac{treasure * k}{N_t} \quad (5)$$

Once it is empty, the house enters the real estate market. In turn, families entering the market are chosen according to the parameter `PERCENTAGE_CHECK_NEW_LOCATIONS`, which is a percentage of the universe of families.

The price of each house is then updated, according to its *size* and *quality* and region.

$$Price_{it} = size * quality * QLI_t \quad (6)$$

The lists of houses available and interested families are sorted according, respectively, to house prices and the amount of families' savings. The first step is for the family to acquire a new ownership. Families that own more than one house decides, at the end of the real estate market procedure, whether to move or not.

Starting from the family with the largest amount, the family will buy the most expensive house it can afford, if any.

Once the matching family-house has been made, the savings of the buying family is deduced of the price paid and the family who previously owned the house receives the same amount. The actual transfer of registering of house ownership is then made, followed by the removal of the house from the sales list.

Once the family has successfully accomplished a purchase, it verifies if it is worth moving or not by checking the quality of the houses and the proportion of employed in the house. Thus, there are three options:

- 1) if the family lives in the best house, but all members of the house are unemployed (or out of the market, such as children and retired members), then the family moves to the second best house;

- 2) if the family does not live in the house with the best quality but at least one person is employed, the family moves into the best house;
- 3) all other options maintain the family in the same house.

Once the decision to move has been made, the family empties the previous house and register the new house, the new address and, eventually, the new region.

4.7 Demographics

Demographics is the module that controls aging, dying and giving births of agents. It is implemented mainly as one function `check_demographics`.

The first step is to check the current year of the simulation (by construction, the first year is 2000). All of the citizens age. Then, according to their gender and month of birth, a death probability, derived from official IBGE data estimates, is evaluated and saved.

Females then evaluate a probability to give birth if they are between 15 and 49 years old. If so, a new agent, child with age zero, is generated and it is set to live within the mother's family.

Then, a number is drawn to evaluate the death of agents. When death occurs, agents are withdrawn from the agents list, from their families and from their firms. For statistical purposes, they are added to a new list called `my_graveyard`.

4.8 Spatial boundness

In the present model, distance is used as one of the criteria for two decisions: when firms choose candidates for employment; and when citizens choose a firm in order to consume its product.

Distance is also used to compute a regional commuting distance, defined as the sum of the commuting distances of all employed citizens living in a given region.

Furthermore, space plays a role in the model when, at the beginning of the simulation, households, firms and citizens are spatially allocated. This allocation is made considering urban and rural proportion for each municipality (see details at section 6.3).

5 RUNNING DIFFERENT SIMULATIONS

The model can run in four different ways: 5.1 Simple; 5.2 Sensitivity analysis; 5.3 Multi-run; and, 5.4 Auto adjustment analysis. The definition of the process of each run is based on the `actual_parameters` module, where each selection is made by means of a combination of variables as logical values (True or False).

These variables are `sensitivity_choice`, `multi_run_simulation`, and `auto_adjust_sensitivity_test`.

Despite all these possibilities, the model's core is the same, but called to run in different ways.

All destinations of outputs from SEAL model are identified by the four different simulation possibilities. Below is the scheme of logical variables and the resulted simulation that is run:

- All logical variables are False: Single run
- Only `sensitivity_choice` is True: `Sensitivity_analysis`
- Only `multi_run_simulation` is True: `Multi_Run_Simulation`
- Only `auto_adjust_sensitivity_test` is True: `Adjust_test`

5.1 Simple run or Test

When the idea is to run a single simulation (a simple test), it is necessary to set all control logical variables (`sensitivity_choice`, `multi_run_simulation`, and `auto_adjust_sensitivity_test`) as False, and call the main module. Automatically, the SEAL model will run a simulation over the selected region and save results accordingly.

The model will save results based on the selection of the saving option in the `actual_parameters` module, so, check carefully the output saving options before running a simulation.

5.2 Sensitivity analysis

In order to run a `sensitivity_choice`, select it as True on `actual_parameters` and call the run from the module

`control_sensitivity_analysis`. This module controls the multiple runs of the main module. As such, the model keeps working as a simple model, but called many times.

The sensitivity analysis is performed over all parameters linked to the economics of the model (ALPHA, BETA, QUANTITY_TO_CHANGE_PRICES, MARKUP, LABOUR_MARKET, SIZE_MARKET, CONSUMPTION_SATISFACTION, PERCENTAGE_CHECK_NEW_LOCATION, TAX_CONSUMPTION), varying the values in between the limits of each one these parameters. For example, for ALPHA [0.01, 1].

Thus, ALPHA will be divided by the number of intervals defined by the user in the `control_sensitivity_analysis` module, set on the parameter `number_of_test_parameter_values` variable as an integer value. Using the same example of ALPHA, if it the parameter `number_of_test_parameter_values` is defined as 6, the values for ALPHA will be: 0.01; 0.208; 0.406; 0.604; 0.802; 1.

However, note that, because of the number of parameters (9) and the example of 6 possible values for each one, a combinatory analysis of 10,077,696 possible combinations is generated. This would take a long time to run. Therefore, we chose to adopt a *ceteris paribus* instance in which each parameter new value is tested while the others remain fixed as default.

Therefore, when the user chooses the number of intervals, the state `True` for sensitivity analysis and runs the simulation calling `control_sensitivity_analysis.py`, the simulation will run all possible runs, as described, and produce a dataset of results, statistics and plots for each parameter. Analysis of the results may indicate how much the results vary based on each different set of parameter values.

5.3 Multi-run

The `Multi-run` simulation is focused on identifying the consistence of the model, varying the random numbers involved on setting the decision-making processes. The idea is to replicate the same model, varying only the random

numbers, so that we can isolate average consistent results, from outlier runs. We aim at reproducing a typical run along with pseudo variance information on its results.

In order to run the `multi_run_simulation`, it is necessary to set the logical variable in `actual_parameters` module to `True` and run the simulation calling the `control_multi_runs_simulations.py`.

In this module, it is necessary to define the number of simulations to run `number_of_runs`.

5.4 Auto-adjustment analysis

The option to use the Auto adjustment analysis aims at verifying the best combination of parameters for the model to produce optimal results. The process is based on three conditions:

The subdivision of each parameter interval based on a number of possibilities defined by the user on `interval_for_values` and the `control_autoadjust_model` module.

The pattern defined as a reference for the model to run the auto adjustment process. The parameters values that produce the maximum GDP and the minimum GINI index at same time in the last month of the simulation are set as default.

The number of times the model will run in order to approximate the best values of parameters that lead to the optimal result is set on `control_autoadjust_model`, specifically setting the variable `times_test_aproximations`, as an integer number.

This structure was defined based on the possible number of combinations for the four most important economic parameters: ALPHA, BETA, MARKUP, and TAX_CONSUMPTION.

In order to restrict a huge number of possibilities, a subdivision method for each parameter was adopted. Thus, in the beginning, the model divides the interval in a number of pieces. For example, for ALPHA and the `interval_for_values` set as 5, the values to be tested will be: 0.01, 0.2575, 0.505, 0.7525,

1 (the other three parameters will remain constant). Such a combination would produce a list of 255 possibilities of parameters.

After the initial approximation, given the respective values of GDP and GINI, the best parameter interval is chosen and a subsequent division of the parameter values is applied. For example, if ALPHA was selected with values 0.505 and 0.7525 for the best combination of GDP and GINI, then the model will produce a new list of five values that lies within the previously set boundaries: 0.505, 0.57, 0.63, 0.69, 0.75; and so on until the number defined of `times_test_aproximations`. Using this approach, the total combinations of model process will be of 1,020 tests.

5.5 Public Policy Test application: example

A final alternative run is to set `alternative0` in the parameters module as `False`. In such a case, the model will run considering, for taxes purposes, all regions within each chosen Population Concentration Area (ACP) [11] as a single region. It is possible to run `alternative0` and `multi_run` simulations together. In such case, the runs alternate between `True` and `False` `alternative0`.

A newer module `control_multi_ACPs_alternate_test` runs the `alternative0` test automatically and plots the differences and median of some indicators for both cases, municipalities as a single entity and municipalities as they are presently.

6 DATA INPUT AND REQUIREMENTS

All dataset used to run the model are available on the internet and come from official Brazilian Agencies websites, especially The Brazilian Institute of Geography and Statistics IBGE.

The data used to run the model is fundamentally from the demographics Census of 2000 and 2010. The data used on the process are divided in Agents, Firms and Government.

6.1 Agents data from IBGE

Agents data come from Demographics Census of 2000 (as a start point):

- Tables of mortality probability by year, age group and gender. The data was interpolated using equal values to generalize it into every year data. Available on: ftp://ftp.ibge.gov.br/Projecao_da_Populacao/Projecao_da_Populacao_2013/tabuas_de_mortalidade_xls.zip.
- Tables of fecundity probability are available by year (between 2000 and 2030), group age (between 15 and 49 years old), and state. Available on: ftp://ftp.ibge.gov.br/Projecao_da_Populacao/Projecao_da_Populacao_2013/projecoes_2013_indicadores_xls.zip.
- IDHM (Human Development Index) by municipalities IPEA/IBGE, Census years (2000, 2010). This measure gives the development conditions for each municipality. Available on: https://docs.google.com/gview?url=http://ivs.ipea.gov.br/ivs/data/rawData/atlasivs_dadosbrutos_pt.xlsx
- Qualification by municipality and divided in years of study. Available on: <http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?z=cd&o=32&i=P&c=2986>

6.2 Firms data from DATAVIVA

- Dataset for number of firms by municipalities, year (from 2002 to 2013), and productive sector. Provided by Ministry of Labor and Employment (MTE) and available on: <http://dataviva.info/pt/data/>.

6.3 Input spatial data

- The spatial dataset is also from IBGE. The model uses the municipalities boundaries maps, available on: ftp://geoftp.ibge.gov.br/organizacao_do_territorio/malhas_territoriais/malhas_municipais/municipio_2010/.
- The urban area boundary definition uses the High Population Concentration Areas (ACPs), defined by IBGE for each metropolitan region of Brazil. These data can be found in: ftp://geoftp.ibge.gov.br/cartas_e_mapas/mapas_do_brasil/sociedade_e_economia/areas_urbanizadas//areas_urbanizadas_

do_Brasil_2005_shapes.zip. These maps represent the spatial delimitation of urban areas (occurrence of populated areas).

- The last demographics data are the urban and rural populations at each municipality, available on IBGE census website (table 200 on IBGE SIDRA database for 2000 and 2010): <http://www.sidra.ibge.gov.br/bda/tabela/listabl.asp?z=cd&o=27&i=P&c=200>.

6.4 Instances generator

The generator module is the one responsible for creating instances of all agents, families, regions, houses and firms. It follows official data and allocates them according to urban and rural proportion, and municipalities population numbers.

Firstly, information on the proportion of urban inhabitants and number of firms by municipality are loaded. Then, information on 2000s Municipal Human Development Index (HDM-I) is loaded.

In order to create regions, the only necessary data is their HDM-I and their respective shapefile, read from 6.3. Such information is then passed on to class `Govern` in `Government`.

All instances are saved in lists that are passed from `time_iteration` to each process. Thus, they are dynamic lists with changing instances.

For time saving purposes, agents for a given spatial configuration and percentage of population are saved in files that can later be loaded, configuring its persistence.

Using the newly created regions, the main function is `create_all`, which creates, within each region, all other mentioned instances.

The first step is to select, from population data for 2000, the number of inhabitants by age group and gender for each municipality. This procedure enables the successive creation of agents for each age group and, alternately, for each gender.

Additionally, each agent gets: a qualification level assigned probabilistically, given municipal data of 2000; a random age that falls within his group age; a small random amount of money; and unique IDs.

The number of families created in each region is proportional to the number of inhabitants with an average given by parameter `MEMBERS_PER_FAMILY` (typically set at 2.5 citizens per family).

Houses are above the number of families, given by parameter `HOUSE_VACANCY` (typically set at 10%).

The number of firms, as stated before, are given by reading of actual data by municipality.

The necessary additional functions are `create_family`, `create_household` and `create_firm`; `allocate_to_family` and `allocate_to_households`; and `get_random_point_in_polygon`.

`Create_family` provides empty family vessels, which are then filled using `allocate_to_family`. For as long as there are agents to be allocated into families for a given municipality, one family and one agent are drawn from the available list and the match is made. There is one check to see if the agent does not belong to any family before allocation. That procedure ensures that the number of agents per family is random and the defined proportion is only valid for the total numbers per municipality.

By construction, the number of families is always smaller than the number of houses available. Hence, all families that have a positive number of members are allocated randomly to an available household using `allocate_to_households`. In such a process which of course happens only before the simulation actually begins the family gets the ownership of that first house they are allocated to. The remaining houses that are empty are then subsequently distributed among all the families for a given municipality.

That implies that, on average, 10% of the families own their own house plus one other empty house that they may make available at the real estate market. Nothing withholds the possibility that luck will provide one family with two or three empty houses.

The `create_household` function is a bit more complex as it needs to abide to some spatial restrictions. Given the urban or rural proportion of the municipality, some of the houses will be set on rural areas (according

to official urban municipal legislation) or on urban areas. That proportion is probabilistic and we do not guarantee that a specific fixed proportion will be urban or rural.

The function that guarantees that actual address falls into the correct urban or rural areas of the map is `get_random_point_in_polygon`.

This function uses `ogr.Geometry`, `geometry().Contains` and `AddPoint` within a given shapefile previously divided into urban and rural areas so that the allocation is correctly determined.

Overall, besides address, the household is also given a random size drawn from the interval (20, 120), a fixed quality, from one to four and an endogenous price value given by Equation 6. Throughout the simulation, size and quality will remain the same, whereas I will be regularly updated, depending on the economic dynamics of the municipality.

In the end, `create_all` returns four lists: `my_agents`, `my_houses`, `my_families` and `my_firms`.⁵

6.5 Parameters and settings

Parameters module contains not only all the parameters of the simulation, but also a lot of settings and decisions. We will describe them below, in order of appearance.

The first parameters is: `PERCENTAGE_ACTUAL_POP` and it establishes the percentage of the population that is going to be used in the simulation. A typical value is 0.01. However, the higher the proportion, the longest the time to complete the simulation. Simple tests can be run with 0.0001.

The second parameter is a decision: `SIMPLIFY_POP_EVOLUTION` and it accepts Boolean values (True or False). When True, the generation of agents is made using age groups. When False, the actual number of inhabitants by each specific age is used in the simulation. However, given smaller municipalities and smaller populations, it becomes inadequate to run a simulation of 1% of the population of 54-year-olds, for example,

when there are only 27 of those. That way, it is easier to reduce the population in bundles of age groups for small municipalities.

`LIST_NEW_AGE_GROUPS` insures that the age groups are malleable. The modeler may define the superior limit of the groups. So far, we have been using [6, 12, 17, 25, 35, 45, 65, 100].

Next comes the decision on region of study. `processing_states` can be any one of Brazilian 27 states or any combination of them, using the two capital letters of its name, such as 'MG' or 'SP'. To select them all, choose 'BR'.

In the sequence, it is possible to choose which Population Concentration Areas (ACPs), established by IBGE in 2005 within each state. ACPs approximate real conurbated metropolitan areas [11]. Thus, they exclude rural, detached municipalities that may be officially part of metropolitan areas but that are not strongly integrated.

Possible ACPs include:

AM, Manaus; PA Belém; AP Macapá; MA São Luís, Teresina; PI Teresina; CE Fortaleza, Juazeiro do Norte/Crato/ Barbalha; RN Natal; PB João Pessoa, Campina Grande; PE Recife, Petrolina/Juazeiro; AL Maceió; SE Aracaju; BA Salvador, Feira de Santana, Ilhéus/Itabuna, Petrolina/Juazeiro; MG Belo Horizonte, Juiz de Fora, Ipatinga, Uberlândia; ES Vitória; RJ Volta Redonda/Barra Mansa, Rio de Janeiro, Campos dos Goytacazes; SP São Paulo, Campinas, Sorocaba, São Jos do Rio Preto, Santos, Jundia, São Jos dos Campos, Ribeirão Preto; PR Curitiba, Londrina, Maringá; SC Joinville, Florianópolis; RS Porto Alegre, Novo Hamburgo/São Leopoldo, Caxias do Sul, Pelotas/Rio Grande; MS Campo Grande; MT Cuiabá; GO Goiânia, Brasília; DF Brasília

According to available data, a starting year has to be selected. For the case of Brazil, the starting data, based on Census data, is the year 2000.

`YEAR_TO_START`

The next selection is the folder where to storage output. It is essential for the running of the program and it has to be set in any new machine.

`OUTPUT_PATH`

Then, a number of settings related to saving,

5. These objects are then saved using `pickles` so that in a subsequent run they can be uploaded and save running time

output, plotting and type of simulation have to be completed with either `False` or `True`.

Another important choice is whether to run with actual municipalities or with merged/changed municipalities. `alternative0` is `True` indicates that boundaries are kept as is. When `False`, municipalities are merged together and collecting of taxes thus benefits all the citizens within a metropolitan region equally.

`sensitivity_choice` refers to the sensitivity analysis (section 5.2); the same is true for `multi_run_simulation` (section 5.3) and `auto_adjust_sensitivity_test` (section 5.4). As said before, a regular run should have all these parameters set to `False`.

`keep_random_seed` is necessary given the fact that `random` module is used in a number of different modules. In order to guarantee that all random numbers are drawn from the same sequence, when needed, a `fixed_seed = random.Random()` is in place.

Some of the settings/control are related to following the simulation as it unfolds, such as: `print_statistics_and_results` and `show_plots_of_each_simulation`.

Another option is `time_to_be_eliminated`, which is related to the amount of data that will enter the plots. The percentage determines the portion to be left out. For example, 0.2 will leave the first 20% of the data out of the plots.

Finally, a number of savings options: `save_plots_figures`, `save_agents_data`, `save_agents_data_monthly`, `save_agents_data_quarterly`, `save_agents_data_annually`, and `create_csv_files`.

The module then initiates the actual simulation running parameters. Those include: `TOTAL_DAYS`, measured in 21 working days a month, meaning that 5.040 days amount to 20 years.

`ALPHA`, which is the productivity factor-decaying exponent and can vary from 0 to 1, being 1 the most productive worker.

`BETA`, which is the consumption exponent and determines the amount the family decides to consume/save. The larger the exponent,

more consumption and less savings.

`QUANTITY_TO_CHANGE_PRICES`, which is the number threshold that firms check against their stocks in order to decide whether demand is high (below the quantity) and thus prices should rise.

`MARKUP`, which is the amount the firm applies when varying prices.

`LABOUR_MARKET`, which is the frequency that the firm makes decisions about entering the market. The higher the parameter, less often.

`SIZE_MARKET`, the number of firms that the consumer check for prices and distances when deciding where to shop.

`CONSUMPTION_SATISFACTION`, this is just a parameter that scales down consumption in order to keep track of accumulated satisfaction given by consumption.

`PERCENTAGE_CHECK_NEW_LOCATION`, the percentage of total families that enter the real estate market each month.

`TAX_CONSUMPTION`, the general consumption tax applied to all sales and collected by municipal government.

The process of rearrange population data is performed after the definition of parameters values. Can be chose to perform a simplification of population when the percentage of population chosen are too small. In the end the parameters module save the files for control process and generate the output `*.txt` data.

6.6 Necessary Python libraries

We run the model in Python 3.4.4.

The following Python libraries are necessary:

```
from numpy import median
from operator
    import methodcaller, attrgetter
from osgeo import ogr
from pandas import read_csv
from timeit import default_timer
ggplot
glob
itertools
matplotlib.pyplot
numpy
os
```

```
pandas
pickle
random
subprocess
sys
```

A simpler way to install the necessary environment is:

```
Install
Anaconda3-2.3.0-Windows-x86_64
```

Then type in Terminal

```
conda install python=3.4.4
conda install pandas=0.18.0
conda install numpy=1.10.4
conda install -c ioos
                geopandas=0.2.0.dev0
pip install ggplot==0.6.8
```

7 OUTPUTS

Output module is called at the end of the month and it is divided into general statistics call `statistics` and regional statistics call `regional_stats`. The former calls `my_statistics` and writes TXT outputs. The information it contains include:

```
actual_month
price_index
GDP_index
unemployment
average_workers
families_wealth
firms_wealth
firms_profit
GINI_index
average_utility
```

The regional statistics include:

```
actual_month
region_id
commuting
get_pop
get_gdp
regional_gini
regional_unemployment
index
GDP_region_capita
```

Some of the methods are detailed in the next subsection. Finally, output module also includes `save_agents_data` and `sum_region_gdp` functions.

7.1 Statistics

The `statistics` class is just a bundle of functions together without a permanent instance of data. Thus, every time `Statistics()` is called, it is initiated anew. The functions include average price of the firms, regional GDP, based on FIRMS' revenues, GDP per capita, unemployment, families' wealth, GINI, regional GINI and commuting information.

`average_price` method checks for every firm in `my_firms` the quantity and price of each product (currently just one). Then it calculates and returns the average.

`calculate_region_GDP` goes through the firms in a given region and adds up the cumulative sold value, inclusive of taxes.

`update_GDP_capita` also goes through the firms in the region, adding up cumulative value sold and then dividing it by current population.

`calculate_unemployment` goes through all agents checking whether they are minor or retired and then classifying those left as employed or unemployed. Then it updates unemployment rate as the ration between unemployed and the total workforce (employed plus unemployed).

`calculate_families_wealth` returns the sum of all families wealth.

`calculate_utility` returns the average of families' average utility. That is, first, the average within each family is calculated and then we calculate the average of such values.

`calculate_GINI` is given by the typical GINI coefficient calculated upon this very families average utility. That is, the GINI is in fact the inequality present among families cumulative consumption. `calculate_regional_GINI` is made exactly the same way but considering only the families residing in a given region (municipality).

`update_commuting` is calculated for every member of each family in each region that is employed.

7.2 Plotting and analysis

The model produces some data and graphs by default in the plotting module and others in accordance to users' choices. The plotting of the model may use different modules.

In all models, the plotting module produces the same set of plots from the `temp_general_%.txt` data. This output data represents the aggregated measures of all regions in the simulation. These measures are: Price index, GDP, Unemployment, Average of number of workers by firms, Families wealth, Firms wealth, Firms profit, GINI index, Average utility. For each one of these measures one plot is produced to allow for a general understanding of the patterns observed in the running model.

The same module produces the firms' plots using data from `temp_firm%.txt`. They represent the Amount produced, Price, Mean and Median of Number employees. They are used to understand the firms' general behavior.

Finally, the plotting module produces regional stats (one value for each measure in each municipality). The measures are: Commuting, Population, GDP of region, GINI by region, Unemployment by municipality, Quality life index by municipality, GDP per capita by municipality.

When the model is in a Multi Run simulation mode, the module `plot_multi_run.py` use the values of `temp_general_%.txt` and stack up each measure (i.e. Unemployment) of all simulations in the same graph. The same process is observed for regional data and agents' data. They are used to produce plots considering each municipality.

7.3 Output data

Output data represents the minimal data necessary to analyze the results of the model. Therefore, all values of simulation for general and regional data are saved by default. These data contain the economics and social measures for the whole selected region `temp_general_%.txt` and

the regional measures (by municipality) of the social and economic measures `temp_regional_stats%.txt`.

The user can choose the scale of agents' data in the files: `temp_firm%.txt`, `temp_agent%.txt` and `temp_house%.txt`, and the time frequency to save such data: monthly: `save_agents_data_monthly`, quarterly: `save_agents_data_quarterly` or annually: `save_agents_data_annually`. Such information is used to produce aggregated or simple plots (in accordance to the type of model running).

7.3.1 Details of TXTs files

When the option to save agents data is set to True, all data is saved and 5 *.TXT files are created.

The files are: 'agent, firm, general, house' and 'regional'. Family information is also available in the house file. They contain no headers, delimiter = , and decimal = .

AGENT contain the following columns: month region_id gender long lat id age qualification firm_id family_id utility distance

FIRM contain the following columns: month firm_id region_id long lat total_balance\$ number_employees total_quantity_in_stock amount_produced price

GENERAL contain the following columns: month price_index gdp_index unemployment average_workers families_wealth families_savings firms_wealth firms_profit gini_index average_utility

HOUSE contain the following columns: month house_id long lat house_size house_price family_id family_savings region_id

REGIONAL contain the following columns: month region_id commuting pop gdp_region regional_gini regional_unemployment qli_index gdp_percapta treasure

The file's names always include the following parameters: "None", alternative0, PERIODICITY_SAVE_DATA, TOTAL_DAYS,

total_pop, SIZE_MARKET, ALPHA,
BETA, QUANTITY_TO_CHANGE_PRICES,
MARKUP, LABOUR_MARKET,
CONSUMPTION_SATISFACTION,
PERCENTAGE_CHECK_NEW_LOCATION, and
TAX_CONSUMPTION.

8 DESIGN CONCEPTS

There are two main drives of the agent-based model here presented. Firstly, it is a restricted adherence to investigate public policies within a spatial continuum. In order to achieve that, there are locational Cartesian attributes for houses and for firms with agents commuting the distance. Further, region space organized in municipalities contain both their actual geodesic boundaries, which determines their action space, but also comprehend the urban rural divide, which enables urban concentration and rural sparseness. Thus, distance explicitly is considered within interaction in the labor market and consumption market. Further, there is a real estate market that is very much dependent on how well the firms in the region are performing. Stronger firms with higher sales increase taxes, which are directly applied to improving estates quality and directly, prices.

Secondly, the model is proposed in a rather flexible and generic way so that it can be constituted as a framework for later public policy analysis. Agents, firms, markets, space, activities are constructed so that newer research questions can be easily adapted into the model, which would provide rapid answering. As it is, for instance, results can be provide for an alteration in a specific tax or a change in a firms strategy to enter the labor market, or a municipality fusion or, yet, in the influence of those changes in mobility patterns, for instance.

Apart from those two specific design concepts, obviously, the model follows the typical agent-based specifications, that is: agent heterogeneity, decision-making based on local variables, and emergence.

Specifically, we can say that the agents in this model adapt in the sense that they change behavior giving observed variables at their environment. For example, firms change

decision-making (and salaries paid) depending on whether they have available balance and profits. Agents decide whether to move into a better quality house depending on the proportion of employed adults in the family.

The model is not intended as a forecasting tool at this stage. The focus is more on setting a framework, learning about complex interactions among different factors across a large spectrum and gaining magnitude of changes insights.

Interactions are plenty. Specifically, they happen in the three proposed markets and within the families (co-responsible for consumption, savings and utility). Indirectly, there is interaction among those agents within the same region.

Stochasticity is present in the model with the use of drawing and random decision-making in a number of procedures. However, a control variable is present that enables the reproduction of the same result. Typically, results should be presented as summaries of a number of simulated runs.

A collective entity that is relevant for the model is that of the family (see section 4.2).

9 FINAL CONSIDERATIONS

This is still a model in progress. However, at this stage, we believe we have most of the framework as intended.

Thus, this research project reveals from a code perspective nearly all aspects of the model. That includes a description of purpose and concept, model scheduling, detailed information and methods on all agents and classes used, the market interactions, data generated and output produced. It also includes information on running the model for multiple runs and explicitly using it as a policy example (section 5.5).

However, we have yet to publish major research studies using such framework, except for [7], [8]. This is exactly the next step: apply the framework to public policy research questions.

Finally, we would like to add that the current working team is short and we would be willing to cooperate with other fellow scientists.

ACKNOWLEDGMENT

The authors would like to thank Institute for Applied Economic Research and especially the Department of Innovation and Infrastructure (DISET) for their continuing support.

REFERENCES

- [1] C. M. Macal, "Everything you need to know about agent-based modelling and simulation," *Journal of Simulation*, vol. 10, no. 2, pp. 144–156, May 2016. [Online]. Available: <http://www.palgrave-journals.com/doi/10.1057/jos.2016.7>
- [2] U. Wilensky and W. Rand, *An Introduction to Agent-Based Modeling*. The MIT Press, 2015. [Online]. Available: <https://mitpress.mit.edu/books/introduction-agent-based-modeling>
- [3] J. M. Epstein, *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press, Dec. 2011.
- [4] J. H. Miller and S. E. Page, *Complex adaptive systems*. Princeton University Press, Mar. 2007.
- [5] J. M. Epstein and R. Axtell, *Growing artificial societies: social science from the bottom up*. Cambridge, MA: Brookings/MIT Press, 1996.
- [6] L. Tesfatsion, "Agent-Based Computational Economics: A Constructive Approach to Economic Theory," in *Handbook of Computational Economics*. Elsevier, 2006, vol. 2, pp. 831–880.
- [7] B. A. Furtado and I. D. R. Eberhardt, "A simple agent-based spatial model of the economy: tools for policy," *Journal of Artificial Societies & Social Simulation*, 2016, arXiv: 1510.04967. [Online]. Available: <http://arxiv.org/abs/1510.04967>
- [8] —, "Da mobilidade metropolitana vinculada a economia : analise a partir de um modelo baseado em agentes," *RADAR*, vol. 43, pp. 9–20, Feb. 2016. [Online]. Available: <http://repositorio.ipea.gov.br/handle/11058/6050>
- [9] V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback, "The ODD protocol: a review and first update," *Ecological Modelling*, vol. 221, no. 23, pp. 2760–2768, 2010. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S030438001000414X>
- [10] B. A. Furtado, P. A. M. Sakowski, and M. H. Tovolli, *Modeling complex systems for public policies*. Brasilia: IPEA, Instituto de Pesquisa Economica Aplicada, 2015.
- [11] IBGE. Ministerio do Planejamento, *Arranjos populacionais e Concentracoes urbanas do Brasil*. Rio de Janeiro: IBGE, 2015.